Metody obliczeniowe w nauce i technice - sprawozdanie 7

Łukasz Jezapkowicz

04.05.2020

1 Korzystając z przykładu napisz program, który:

1. Jako parametr pobiera rozmiar układu równań n

2. Generuje macierz układu $A(n \times n)$ i wektor wyrazów wolnych $b(n)$

3. Rozwiązuje układ równań

4. Sprawdza poprawność rozwiązania (tj., czy $Ax = b$)

5. Mierzy czas dekompozycji macierzy - do mierzenia czasu można skorzystać z przykładowego programu dokonującego pomiaru czasu procesora spędzonego w danym fragmencie programu.

6. Mierzy czas rozwiązywania układu równań

1.1 Program

Program został przeze mnie napisany w języku $C++$. Pokażę teraz kolejne kawałki programu tłumacząc ich działanie.

```
srand(time(NULL));
struct rusage t0,t1,t2;

/* getting the matrix A size */
cout << "Give the A matrix size: ";
int n;
cin >> n;

/* creating and generating matrix values */
double *A = new double[n*n];
double *C = new double[n*n];
for (int i=0; i<n*n; i++)
    A[i] = rand() / 1e6;
double b[n];
for (int i=0; i<n; i++)
    b[i] = rand() / 1e6;

/* printing generated matrixes */
cout << "\nMatrix A:\n";
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        C[i*n+j] = A[i*n+j];
        cout << left << setw(5) << A[i*n+j] << " ";
    }
    cout << endl;
}
```

Na początku ustawiam *seed* zgodnie z aktualnym czasem, dzięki czemu generowane macierze zawsze będą się różniły. Następnie pobieram wymiar macierzy n, tworzę odpowiednie macierze i wypełniam je losowymi liczbami zmiennoprzecinkowymi. Warto dodać, że tworzę dodatkowo macierz C, która jest kopią macierzy A - będzie ona potrzebna do weryfikacji poprawności wyniku. Ostatnim krokiem jest wypisanie wygenerowanej macierzy A.

```
cout << "\nMatrix B:\n";
for (int i=0; i<n; i++)
    cout << b[i] << "\n";

/* creating gsl matrixes */
gsl_matrix_view A_m = gsl_matrix_view_array(A,n,n);
gsl_vector_view b_m = gsl_vector_view_array(b,n);
gsl_vector *x = gsl_vector_alloc(n);

/* solving the equation Ax = b */
int s;
gsl_permutation *p = gsl_permutation_alloc(n);
getrusage(RUSAGE_SELF,&t0);
gsl_linalg_LU_decomp(&A_m.matrix,p,&s);
getrusage(RUSAGE_SELF,&t1);
gsl_linalg_LU_solve(&A_m.matrix,p,&b_m.vector,x);
getrusage(RUSAGE_SELF,&t2);

/* printing the result */
cout << "\nResult x:\n";
gsl_vector_fprintf(stdout,x,"%g");
time_usage(&t0,&t1,"Decomposition time:");
time_usage(&t1,&t2,"Solving time:");
```

Następnym krokiem jest wypisanie wygenerowanej macierzy b. Kolejno tworzę odpowiednie struktury zgodne z biblioteką GSL, tworzę dekompozycję LU oraz rozwiązuje układ (mierząc czas odpowiednich operacji) i wypisuję znalezione rozwiązanie oraz czas trwania odpowiednich operacji. Do mierzenia czasów użyłem udostępnionego rozwiązania, lekko je modyfikując. Funkcja *time_usage* widoczna jest poniżej.

```
int time_usage(struct rusage *ru0, struct rusage *ru1, string msg){

    double utime = 0, stime = 0, ttime = 0;

    utime = (double) ru1->ru_utime.tv_sec
        + 1.e-6 * (double) ru1->ru_utime.tv_usec
        - ru0->ru_utime.tv_sec
        - 1.e-6 * (double) ru0->ru_utime.tv_usec;
    stime = (double) ru1->ru_stime.tv_sec
            + 1.e-6 * (double) ru1->ru_stime.tv_usec
        - ru0->ru_stime.tv_sec
            - 1.e-6 * (double) ru0->ru_stime.tv_usec;
    ttime = stime + utime;
    int seconds = ttime;
    ttime -= seconds;
    int milliseconds = ttime * 1e3;
    ttime -= milliseconds*1e3;

    cout << msg << " Seconds: " << seconds << " Milliseconds: " << milliseconds << " Microseconds: " << ttime * 1e6 << endl;
}
```

```
/* printing the A*x */
gsl_matrix_view C_m = gsl_matrix_view_array(C,n,n);
gsl_vector *Ax = gsl_vector_alloc(n);
gsl_blas_dgemv(CblasNoTrans,1.0,&C_m.matrix,x,0.0,Ax);
cout << "\nA*x:\n";
gsl_vector_fprintf(stdout,Ax,"%g");

/* checking if A*x = y */
bool proper = true;
for (int i=0; i<n; i++)
    if(abs(Ax->data[i]-b_m.vector.data[i]) > 1e-6)
        proper = false;

if (proper)
    cout << "\nThe result x is correct.\n";
else
    cout << "\nThe result x is incorrect.\n";

/* freeing memory */
delete[] A;
delete[] C;
gsl_permutation_free(p);
gsl_vector_free(x);
gsl_vector_free(Ax);

return 0;
```

Kolejnym krokiem jest stworzenie wektora $A * x$ oraz sprawdzenie czy jest on równy (z pewnym błędem) naszej macierzy b. Wektor $A * x$ tworzę przy pomocy funkcji gsl_blas_dgemv (d - double precision, ge - general matrix, mv - matrix-vector). Następnie wypisuję znaleziony wektor oraz sprawdzam czy wartości $A * x$ oraz b są równe z pewnym błędem (tu przyjąłem 10^{-6}). Wypisuję informację czy rozwiązanie jest prawidłowe. Na koniec zwalniam pamięć oraz konczę działania programu.

1.2 Test działania

```
Give the A matrix size: 4

Matrix A:
1269.62 2086.23 96.0763 1448.34
14.7616 515.435 54.5266 163.618
1141.66 701.821 1753.58 799.804
1713.05 1528.72 603.446 616.67

Matrix B:
358.573
849.563
1566.8
1744.41

Result x:
-0.68143
2.32912
1.59813
-2.61603
Decomposition time: Seconds: 0 Milliseconds: 0 Microseconds: 10
Solving time: Seconds: 0 Milliseconds: 0 Microseconds: 10

A*x:
358.573
849.563
1566.8
1744.41

The result x is correct.
```

```
Decomposition time: Seconds: 0 Milliseconds: 0 Microseconds: 11
Solving time: Seconds: 0 Milliseconds: 0 Microseconds: 7
```

Czas działania dla $n = 10$

```
Decomposition time: Seconds: 0 Milliseconds: 0 Microseconds: 228
Solving time: Seconds: 0 Milliseconds: 0 Microseconds: 23
```

Czas działania dla $n = 100$

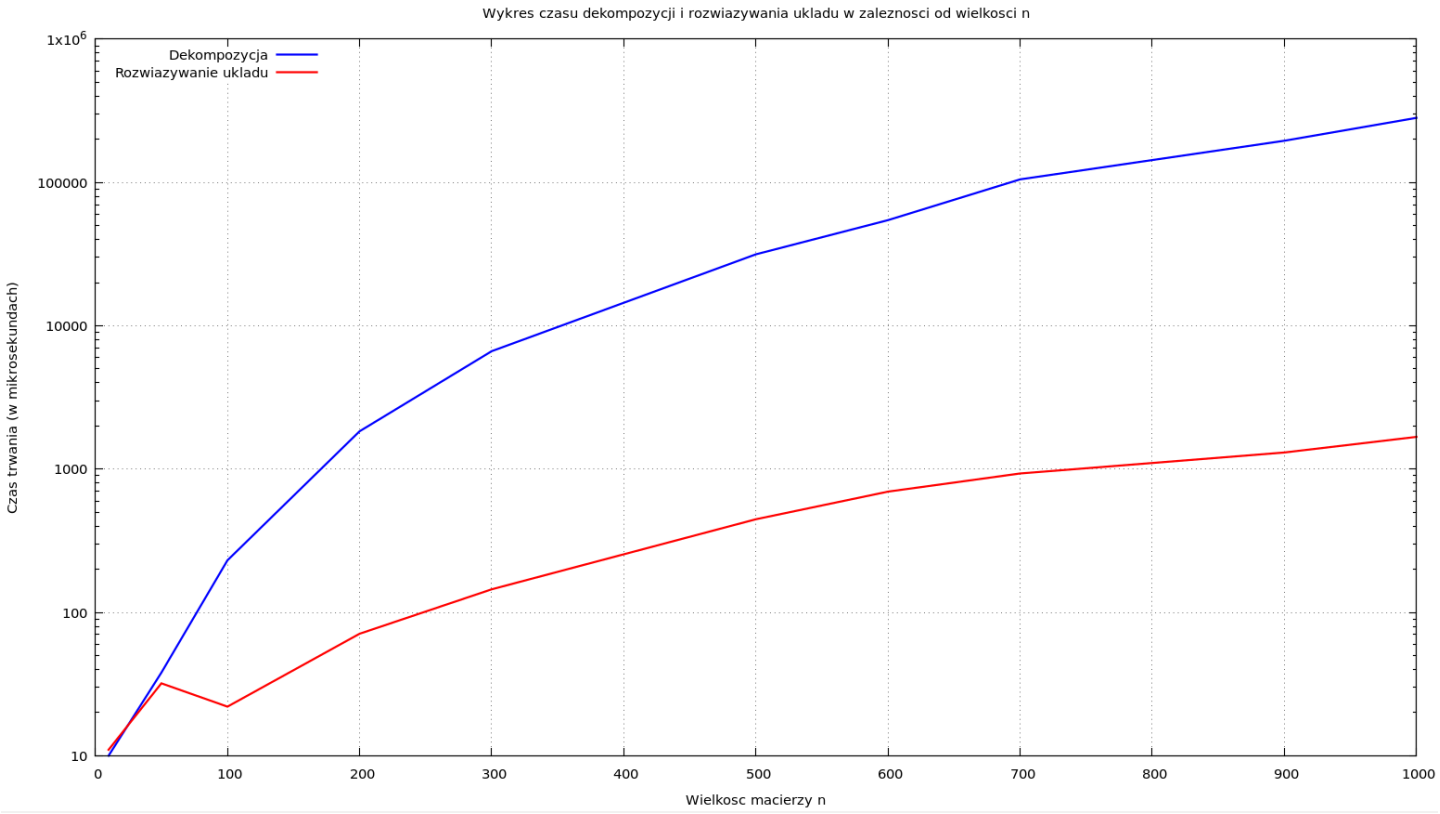
```
Decomposition time: Seconds: 0 Milliseconds: 266 Microseconds: -2.66e+11
Solving time: Seconds: 0 Milliseconds: 1 Microseconds: -9.99998e+08
```

Czas działania dla $n = 1000$

Jak widać, czas działania dosyć szybko zwiększa się wraz z wielkością danych wejściowych. Widać również, że czas dekompozycji LU zajmuje o wiele więcej czasu niż rozwiązanie układu.

2 Narysuj wykres zależności czasu dekompozycji i czasu rozwiązywania układu od rozmiaru układu równań. Wykonaj pomiary dla 10 wartości z przedziału od 10 do 1000.

Wybrane przeze mnie wartości z przedziału od 10 do 1000 to 10,50,100,200,300,500,600,700,900,1000. Wykres narysowany jest przy pomocy *GNUPLOT*.



Na wykresie zastowałem skalę logarytmiczną dla osi *OY*. Czas dekompozycji jest jak widać o wiele większy niż czas rozwiązywania układu po dekompozycji. Mimo to warto stosować metodę dekompozycji *LU*, ponieważ czas całkowity rozwiązywania układu jest korzystniejszy niż dla innych metod (np. Gaussa).

3 Wniosek

Używanie biblioteki *GSL* do rozwiązywania układów równań liniowych jest dosyć łatwe oraz efektywne. Warto więc stosować tą bibliotekę do rozwiązywania problemów numerycznych w połączeniu z *GNUPLOT*, dzięki któremu możemy łatwo zobrazować wyniki.